- Eine weitere bedingte Schleife kann in Java mit dem Schlüsselwort for definiert werden:

```
for (<Initialisierung>; <Bedingung>; <Update>)
  <Anweisungsfolge>
```

- Alle drei Bestandteile im Schleifenkopf sind Ausdrücke (nur `<Bedingung>` muss vom Typ **`boolean`** sein).
- *Vorsicht*: Dieses Konstrukt ist keine klassische gezählte Schleife (auch wenn es for-Schleife genannt wird).

- Der Ausdruck `<Initialisierung>`
  - wird einmal vor dem Start der Schleife aufgerufen
  - darf Variablendeklarationen mit Initialisierung enthalten (um einen Zähler zu erzeugen); diese Variable ist nur im Schleifenkopf und innerhalb der Schleife (`<Anweisungfolge>`) sichtbar aber nicht außerhalb
  - darf auch fehlen
- Der Ausdruck `<Bedingung>`
  - ist ähnlich wie bei den While-Konstrukten die Testbedingung
  - wird zu Beginn jedes Schleifendurchlaufs überprüft
  - die Anweisung(en) `<Anweisungsfolge>` wird (werden) nur ausgeführt, wenn der Ausdruck `<Bedingung>` den Wert **`true`** hat
  - kann fehlen (dies ist dann gleichbedeutend mit dem Ausdruck **`true`**)
- Der Ausdruck `<Update>`
  - verändert üblicherweise den Schleifenzähler (falls vorhanden)
  - wird am Ende jedes Schleifendurchlaufs ausgewertet
  - kann fehlen

- Eine gezählte Schleife wird in Java wie folgt mit Hilfe der for-Schleife notiert:

```
for (<Zaehler>=<Startwert>;
     <Zaehler> <= <Endwert>;
     <Zaehler> = <Zaehler> + <Schrittweite>)
<Anweisungsfolge>
```

- Beispiel: Fakultät

```
public static int fakultaet05(int n)
{
  int erg = 1;
  for(int i=1; i<=n; i++)
  {
    erg = erg * i;
  }
  return erg;
}
```

4.4 Imperative Algorithmen

4.4.3 Verzweigung und Iteration

- In Java gibt es Möglichkeiten, die normale Auswertungsreihenfolge innerhalb einer **`do`**-, **`while`**- oder **`for`**-Schleife zu verändern.
- Der Befehl **`break`** beendet die Schleife sofort. Das Programm wird mit der ersten Anweisung nach der Schleife fortgesetzt.
- Der Befehl **`continue`** beendet die aktuelle Iteration und beginnt mit der nächsten Iteration, d.h. es wird an den Beginn des Schleifenrumpfes "gesprungen".
- Sind mehrere Schleifen ineinander geschachtelt, so gilt der **`break`** bzw. **`continue`**-Befehl nur für die aktuelle (innerste) Schleife.

- Mit einem *Sprungbefehl* kann man an eine beliebige Stelle in einem Programm "springen".
- Die Befehle **`break`** und **`continue`** können in Java auch für (eingeschränkte) Sprungbefehle benutzt werden.
- Der Befehl **`break`** `<label>;` bzw. **`continue`** `<label>;` muss in einem Block stehen, vor dem die Marke `<label>` vereinbart ist. Er bewirkt einen Sprung an das Ende dieses Anweisungsblocks.
- Beispiel:

```
int n = 0;
loop1:
for (int i=1; i<=10, i++)
{
  for (int j=1, j<=10, j++)
  {
    n = n + i * j;
    break loop1;
  }
}
System.out.print(n); // n = 1
```

Der Befehl **`break`** `loop1;` erzwingt den Sprung an das Ende der äußeren **`for`**-Schleife.

Anmerkung:
Durch die Sprungbefehle (vor allem bei Verwendung von Labeln) wird ein Programm leicht unübersichtlich und eine Korrektheitsüberprüfung wird schwierig. Wenn möglich, sollten Sprungbefehle daher vermieden werden.

- Sog. *unerreichbare Befehle* sind Anweisungen, die (u.a.)
  - hinter einer **`break`**- oder **`continue`**-Anweisung liegen, die ohne Bedingung angesprungen werden,
  - in Schleifen stehen, deren Testausdruck zur Compile-Zeit **`false`** ist.
- Solche unerreichbaren Anweisungen sind in Java nicht erlaubt, sie werden vom Compiler nicht akzeptiert.
- Einzige Ausnahme sind Anweisungen, die hinter der Klausel

```
if (false)
```

stehen. Diese Anweisungen werden von den meisten Compilern nicht in den Bytecode übernommen, sondern einfach entfernt. Man spricht von *bedingtem Kompilieren*.